排序应用

2019年4月4日 16:12

一. 离散化
 1. 离散化：将无穷大集合中若干元素映射到有限离散集合上
    i. 最简单的方法是对数组排序，去重，将下标与数值本身视作映射
    ii. std::unique(bgn,ed)负责将[bgn,ed)之间，相邻的重复元素后移，即不改变容器本身的长度，且容器最后的值可能是原容器中任意位置的值，因此这个函数会**返回理论上的新ed迭代器**
    iii. **只有保证重复元素都相邻，才能保证去重**，因此一般事先std::sort
    iv. 如果要删除重复元素，可以调用容器.erase(unique返回值,容器.end())
    v. 容器内的**数据量m=unique返回值-容器.begin()**
 2. void descrete(){//离散化（假设数据在a[1]~a[n]，m初值0）
```
std::sort(a+1,a+1+n);
a[0]=a[1]-1;
for__(i,1,n)
    if(a[i]!=a[i-1])
        a[++m]=a[i];}
```
 3. void descrete(){//离散化
```
std::sort(a+1,a+1+n);
newn= std::unique(a+1,a+1+n)-a-1;}
```
 4. inline int query(int x){//查找x的序号
```
return std::lower_bound(a+1,a+1+newn,x)-a;}
```

二. 离散化例题
 1. 扫描线拐点：n行，每行lrh表示[l,r]区间有一个高为h的正方形，可重叠
    i. 因为l和r的范围极大，不能直接作为下标，需要离散化，为了保存h的信息，开一个struct
    ii. 利用multiset自动排序，实现直接找最大值
    iii. 注意multiset的erase(t)是删除所有值=t的元素，返回删除个数
    iv. const int MN = 200005;
```
struct ST{
    int x, h; //坐标，高度
    bool l;          //是不是左端点
    bool operator<(const ST r)const{
        return x < r.x;
    }       //按横坐标大小排序
}s[2*MN];
```
    v. int n;
```
scanf("%d",&n);
int h;
for_(i,0,n){
    scanf("%d%d%d",s+i,s+n+i,&h);
    s[i].h= s[i+n].h= h;
    s[i].l= 1;}
int N= n*2;
sort(s,s+N);
```

```
int x;
int oldh=0, newh=0;
multiset<int> ms;          //当前存在的货物的高度
for_(i,0,N){
        x= s[i].x;
        while(s[i].x==x){
                if(s[i].l)
                        ms.insert(s[i].h);
                else{
                        int t= ms.erase(s[i].h);   //删去所有该值的数
                        for_(j,1,t)          //把一并删去的t-1个数再加回去
                                ms.insert(s[i].h); }
                ++i;}
        --i;        //否则for循环会跳过这个i
        if(ms.empty())
                newh=0;                    //没货了
        else
                newh=*(--ms.end());
        if(newh!=oldh)
                printf("%d %d\n%d %d\n",x,oldh,x,newh);
        oldh= newh;}
```

三. 逆序对

1. i<j且a[i]>a[j]时称a[i]与a[j]构成逆序对
2. 稳定归并排序顺便求逆序对：a[j]<a[i]时，[1,m]都与j构成逆序对
3. 注意如果a是有重序列，一定要把a[i]==a[j]这个情况算在让t[k]=a[i++]里
4. void merge(int l,int r){

```
        if(l==r)
                return;
        int m=(l+r)>>1;
        merge(l,m);
        merge(m+1,r);
        int i=l,j=m+1;
        for__(k,l,r)
                if(j>r || i<=mid&& a[i]<=a[j])
                        t[k]=a[i++];
                else
                        t[k]=a[j++],
                        cnt+=mid-i+1;
        for__(k,l,r)
                a[k]=t[k];}
```

5. 冒泡排序：每交换一次就++逆序对一次

四. 三维偏序

1. f(i)=三维数据全<=a[i]的a的数量
2. 为了方便统计重复数据，先去重，每次在给答案+=重复次数-1
3. 先以一维为第一关键字，二维为第二关键字，三维为第三关键字排序，确保每次二分能保证把三维全都小于a[i]的并到a[i]左边
4. 之后以二维为第一关键字，三维为第二关键字做cdq分治
5. 每次cdq分治，按第三维插入值域树状数组，对每个a[i]计算值域树状数组上<=第三维的数据量即可实现

```
struct BIT{
	int N,v[MN<<1];
	void init(int nn){ N=nn; for__(i,0,N) v[i]=0; }
	ll sum(int x){ ll s=0; for(; x>0; x-= x&-x) s+=v[x]; return s; }
	void add(int x,int d){ for(; x<=N; x+= x&-x) v[x]+=d; }
}bit; //值域树状数组统计z的排名
struct Node{
	int x,y,z,ans,cnt; //三维，小于等于次数，重复点次数
	bool operator!=(const Node&t){
		return x!=t.x || y!=t.y || z!=t.z;
	}
}a[MN],o[MN];
bool cmpx(Node &l,Node &r){
	return l.x!=r.x? l.x<r.x: (l.y!=r.y? l.y<r.y: l.z<r.z) ;
}
bool cmpy(Node &l,Node &r){
	return l.y!=r.y? l.y<r.y: l.z<r.z;
}

void cdq(int l,int r){    //[l,r)
	if(l+1==r) return;
	int mid= l+r >>1;
	cdq(l,mid), cdq(mid,r);
	sort(a+l,a+mid,cmpy), sort(a+mid,a+r,cmpy);
	int i=mid, j=l;     //右区间指针i，左区间指针j
	for(; i<r; ++i){    //为每个右区间数据统计左区间的总贡献量
		while(a[j].y<=a[i].y && j<mid)
			bit.add(a[j].z,a[j].cnt), ++j;
		a[i].ans+=bit.sum(a[i].z);
	}
	for_(t,l,j) bit.add(a[t].z,-a[t].cnt); //清空左区间的计数
}

int n,k,newn,ans[MN<<1];  //元素数，值域端点，去重后元素数
inline void solve(){
	n=read(), k=read(), bit.N=k;
	for_(i,0,n) o[i].x=read(), o[i].y=read(), o[i].z=read();
	sort(o,o+n,cmpx);  //先按第一维排序
	int cnt=0;   //当前元素重复次数
	for_(i,0,n){
		++cnt;
		if(o[i]!=o[i+1]) a[newn]=o[i], a[newn++].cnt=cnt, cnt=0;
	}
	cdq(0,newn);
	for_(i,0,newn) ans[ a[i].ans+a[i].cnt-1 ]+=a[i].cnt;
	for_(i,0,n) write(ans[i]), putchar('\n');
}
```